DESIGN AND OPERATION OF TOPEX/POSEIDON AUTONOMOUS MANEUVER EXPERIMENT (TAME)

Parag Vaze, Tooraj Kia, Allan Klumpp, Jeff Mellstrom
Jet Propulsion Laboratory
4800 Oak Grove Dr.
Pasadena,CA 91109
(909)902-5664
Parag.Vaze@jpl.nasa.gov

Abstract—Spacecraft autonomy is becoming an important aspect of the design and operation of future space missions. While science objectives continually increase, budgets for development and operations remain fairly flat. Implementing spacecraft autonomy may help in achieving these goals, while reducing operations costs.

Autonomy algorithms have been developed and tested using ground simulations, however the key test lies in the flight performance of these techniques. In order to test these algorithms, the TOPEX/POSEIDON Autonomous Maneuver Experiment (TAME) has been designed, implemented and executed in flight. TAME is an experiment to provide the necessary algorithms for planning and executing attitude maneuvers and a thrusting Orbital Maintenance Maneuver (OMM) autonomously. This experiment not only provides the challenge of developing the autonomy algorithm but also implementing them on an operational satellite that is not designed to accommodate autonomous attitude or propulsive maneuvers. This paper describes the general experiment design with special emphasis on the implementation difficulties and considerations required for implementation on a flight operational satellite. Although the task of implementing autonomy on an existing flight program provides a challenge, it is achievable. This may provide an attractive solution to limit operations costs and increase the operations flexibility to achieve additional science objectives for not only future missions but also for existing flight programs.

TABLE OF CONTENTS

1. INTRODUCTION

Spacecraft autonomy is becoming increasingly important in planning future space missions and operating existing missions with a reduced ground control presence[1]. Missions will continue to become more ambitious, scientifically and technically, and will demand more autonomy to accomplish complex tasks in uncertain environments and in close proximity to celestial bodies. Affordability is now an additional primary driver of autonomous missions. Sponsors are calling for smaller missions with greatly reduced ground control and operation. Spacecraft with highly autonomous, goal directed systems are required to meet these new challenges. In addition to reducing the mission operations cost, autonomous systems will enable science objectives not possible with current spacecraft architectural designs. Autonomous maneuver planning and implementation is a key technology for future missions.

This paper describes the software design and in-flight implementation of the TOPEX/POSEIDON Autonomous Maneuver Experiment[2,3] (TAME) . It describes TAME's architecture, its primary module (the *planner*), associated software, flight implementation strategy and the results of the final flight test. This experiment will provide proof-of-concept technology for an important area of on-board autonomy.

TAME provides the algorithms necessary for autonomously planning attitude maneuvers that execute an Orbital Maintenance Maneuver (OMM) without violating constraints. An OMM is accomplished by pointing the spacecraft thruster in the direction of the requested velocity increment (ΔV), and thrusting until the requested ΔV is imparted. In the TOPEX case, the velocity increment is always imparted in the direction of the current orbital velocity vector, to correct for drag. Therefore ΔV direction is never commanded.

The planner and its internal database resides on an existing satellite processor, along with several auxiliary modules. Some modules have their own databases. The databases contain certain satellite and orbit constants as well

as tables of mission constraints. Upon receiving a maneuver ΔV and other data from ground commands, TAME requests tank pressures from the satellite's On-Board Computer (OBC). TAME uses the ΔV and tank pressures for computing burn duration. The planner then generates a *maneuver plan* for turning from cruise attitude to burn attitude and back, while avoiding violations of geometric, power, and thermal constraints en route. The *sequence generator*, merges the maneuver plan with predefined *OMM templates*, yielding an *OMM sequence* that avoids violations of other types of constraints, such as the timing and order of commands. The OMM sequence, which includes the commands to reconfigure and to condition the satellite and its components, is transmitted to the OBC for execution.

This technology demonstration is providing data to perform cost/benefit analysis for trades between flight- and ground-based spacecraft mission operations. It is also providing approaches for new paradigms in system architecture, ground commanding, and test and verification that are necessary for designing highly autonomous event-driven flight systems.

The system originally planned (described in references 2 and 3) would not fit in the 1750A flight computer. Therefore, some on-board capabilities were eliminated, including spacecraft and solar ephemerides.

2. TOPEX/POSEIDON MISSION

The TOPEX/Poseidon Satellite, herein referred to as TOPEX (Ocean Topology Experiment) was launched on August 10, 1992, from the Kourou Space Center in French Guyana. The satellite was launched into a nominal circular orbit with an altitude of 1336 Km and an inclination of 66 degrees. The TOPEX is a remote sensing mission with the primary science objective of providing sea surface altimetry from space[4]. The TOPEX/Poseidon program is jointly sponsored by The National Aeronautics and Space Administration (NASA) and Centre National d'Etudes Spatiales (CNES). This joint U.S./French mission combines each country's space ocean research missions. The Jet Propulsion Laboratory (JPL) manages the TOPEX mission for the NASA office of Space Sciences Application. JPL is also responsible for the day to day operation of the satellite. The Toulouse Space Laboratory manages the Poseidon for CNES. TOPEX was slated for a prime mission of three years, which was completed in September 1995. TOPEX is currently in its forth year of an extended operation approved by NASA.

The primary science objective of the TOPEX satellite is to provide highly accurate measurements of the sea surface elevation over all of the ocean basins. The primary science requirement is to provide geocentric measurement of the global ocean sea level accurate to $\pm$ 14 cm with a precision of $\pm$ 2.4 cm along track. These requirements necessitated a frozen orbit that provides a fixed ground track every 10 sidereal days (127 orbits)[5]. To maintain the frozen orbit, the satellite occasionally performs a small thrusting maneuver referred to as an Orbit Maintenance Maneuver (OMM) or a burn.

3. TAME OBJECTIVE

An OMM consists of three phases: a planning phase, an implementation phase, and an execution phase. In the planning phase, the spacecraft orbit is determined, and the required velocity correction (the maneuver ΔV) is computed. In the implementation phase, a constraint free attitude maneuver design is generated based upon propulsion, power, thermal and telecommunications inputs. The attitude maneuver is then converted into a sequence of low-level commands (an OMM sequence), which are loaded into a flight computer for execution. After the OMM sequence is verified, a proceed command is issued to initiate the execution phase. Traditionally, the first two phases are carried out on the ground. TAME's objective was to develop capabilities for carrying out the implementation phase autonomously, in the flight computer.

For TOPEX, the current implementation phase involves a labor-intensive approach that requires an iterative exchange of data between the different sub-system groups before a solution is achieved as shown in Figure 1. The typical maneuver design requires 3 to 4 iterations to design a constraint free maneuver implementation design. This process not only requires iteration but also involves the interaction between several different teams of people who utilize separate tools to generate design inputs. This methodology requires substantial labor, time and computing resources to support this design function. This approach may be favored in situations where the project requires few maneuvers or special designs for each maneuver case.

In a fully autonomous approach, all three phases would be carried out in flight without any ground intervention. Although technically feasible for most modern satellites, this level of autonomy was deemed as inappropriate for TAME. TOPEX flight computers, the OBC and the 1750A, simply do not have sufficient unused capacity to support completely autonomous maneuver functions. Since the execution phase of an OMM has long been autonomous, it was decided that the next logical step would be to convert the implementation phase to an autonomous operation. With TAME, the planning phase remains a ground activity.

The objective for TAME is to replace the functions currently performed on the ground as shown inside the dashed box on Figure 1. The autonomous maneuver process that embeds the ground functions on-board the satellite is shown in Figure 2. The planning phase is carried

out on the ground by a navigation team; the resulting maneuver ΔV and other data are uplinked to the spacecraft. The implementation phase is carried out in flight; the resulting OMM sequence is downlinked to the ground for verification and to the OBC for execution. The execution phase is carried out in flight following receipt of a command to proceed.

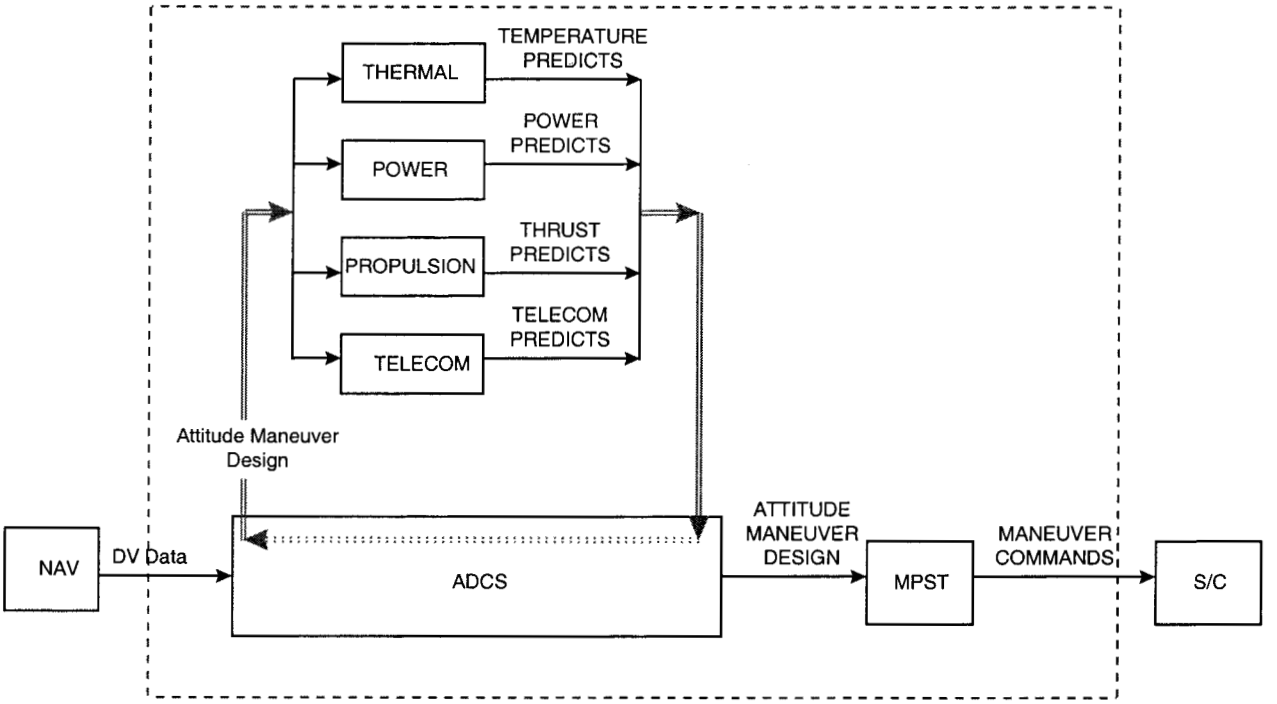

Figure 1: TOPEX/Poseidon Maneuver Process

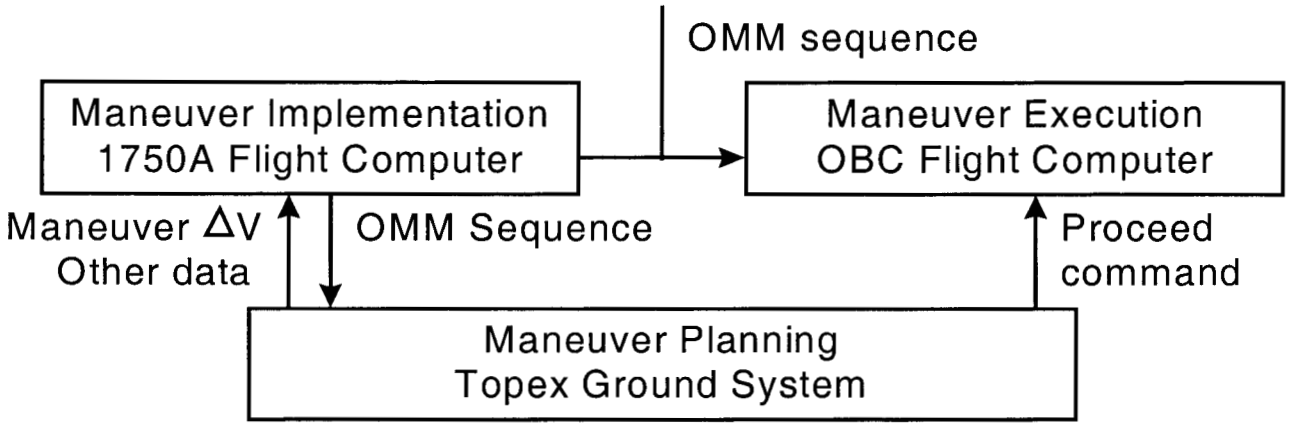

Figure 2: TOPEX/Poseidon Autonomous Maneuver process

4. TAME ARCHITECTURE

The TAME architecture is developed around the challenge of implementing an autonomous maneuver strategy on an existing satellite with hardware and software not intended for such functionality. The main hardware obstacle was the lack of computing resources on the On-Board Computer (OBC). This problem was overcome by utilizing another on-board processor (1750A) utilized by the Global Positioning System Demonstration Receiver (GPSDR). Although the 1750A offered additional computing resources, a two-way communications channel did not exist between the two flight computers. Designing and implementing a robust and stable two-way communications channel required the development of a completely new communications technique as described later in the paper.

The other key challenge was to develop a modular and reusable implementation that can easily be customized for other missions. Figure 3 shows TAME's modular architecture. In this architecture, the OMM sequence is similar to a ground-generated OMM sequence. Thus the OBC utilizes existing maneuver routines to execute the sequence.

At the heart of the process is the planner. The planner processes OMM requests and generates the constraint free OMM design. Inputs to the planner are burn, window, search, and turn parameters. Maneuver ΔV is among the burn parameters. The planner utilizes a set of physical models that generate predicts for each of the relevant on-board sub-systems (Power, Thermal, Propulsion, ADCS). The propulsion model computes burn duration using maneuver ΔV from the planner and tank pressures from the OBC. Each physical model was developed using a simplified version of the physical model used by the existing ground software. Sufficient margin was used to accommodate errors introduced by the simplification. The interface between the planner and the physical models has been standardized such that a minimal impact would occur if model modifications or replacement was required for other mission designs.

The planner outputs are passed to the Command Translator and the Sequence Generator (SEQGEN) for conversion into the mission specific command format. The output of SEQGEN is transferred to the OBC for execution. Other inputs and other interfaces between modules are described later.

Communication between Flight Computers

Communication between the OBC and 1750A provided another challenge to the TAME design. Figure 4 graphically shows the data transfer architecture between the three major interfaces; ground and OBC, ground and 1750A, and 1750A and OBC. Communication is based on the existing TOPEX command and telemetry architecture with major modifications to the format of the commands and telemetry. Special attention was placed upon making the communications interfaces robust and independent from existing flight command and telemetry interfaces. Modifications to the existing command and telemetry predicated the need for developing or modifying ground software tools for interpreting the telemetry or generating the commands. These support tools required substantial efforts and played an essential role in the success of TAME.

To establish the 1750A interface with the ground, TAME telemetry was embedded within the existing 1750A telemetry. The TAME telemetry provided data regarding the health of the 1750A, the execution status, command confirmation, and downlink of the maneuver design. A similar strategy was utilized for commanding where the TAME commands were embedded within existing 1750A command channels.

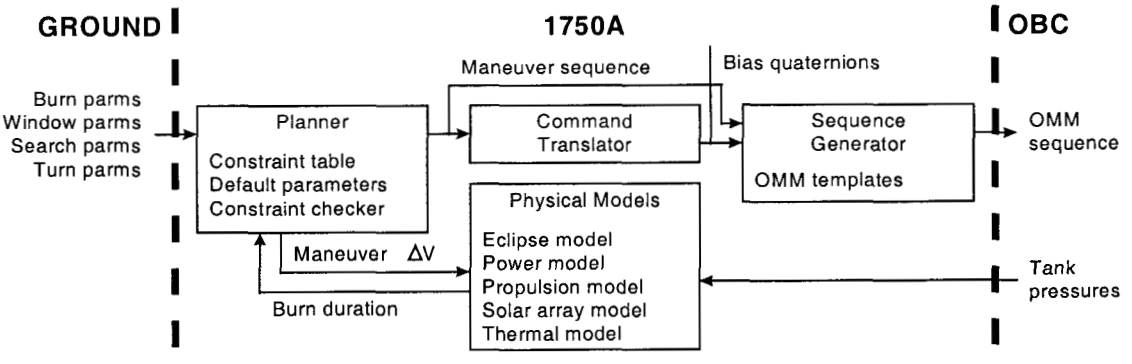

Figure 3: TAME Modular Architecture

The C.U. is the focal point for command and telemetry handling on TOPEX. All commands and telemetry are processed and routed via the C.U. Since the OBC has access to the complete satellite telemetry stream via the C.U., this allowed for data transfer from the 1750A to the OBC. For TAME, 1750A telemetry is modified by embedding spacecraft commands, to be executed by the OBC, in place of normal telemetry. In this experiment, the OBC software modification allows the OBC to capture, interpret, and verify the spacecraft commands from the 1750A. After extracting these spacecraft commands from 1750A telemetry, the OBC stores the TAME commands in the Absolute Time Command Buffer (ATCB) for execution at a later time. This completes the TAME maneuver implementation phase. TAME does not change the maneuver execution phase.

To communicate with the 1750A, the OBC utilized its existing capability to generate and issue satellite commands. The OBC and ground interface utilized existing command and telemetry channels with minor modifications to the command and telemetry formatting and structure. This strategy though conceptually simple, required a complex communications protocol to allow for command and telemetry errors, bus errors, and timing and synchronization issues.

1750A SOFTWARE ARCHITECTURE

For each of the purposes of the experiment, the 1750A computer in which TAME resides acts as a co-processor to the OBC. The 1750A performs all the calculations for planning the requested maneuver and generates a complete sequence to implement the planned maneuver. The OBC receives and stores the maneuver sequence of absolute timed commands from the 1750A. The sequence is then interpreted and executed.

The software on the 1750A computer is substantially original. While it was developed specifically for the TAME experiment, the underlying software architecture and command and telemetry interfaces were inherited from the pre-TAME application. The software on the OBC, on the other hand, is substantially unchanged. A patch was made to the existing software to accommodate an interactive interface with the 1750A, to receive and process a sequence generated by the 1750A, and to allow detailed ground control over execution of the autonomously generated sequence.

The inherited architecture of the 1750A software consists of a *main* program that spawns two processes referred to as the *high-* and *low-priority tasks* (see Figure 6).

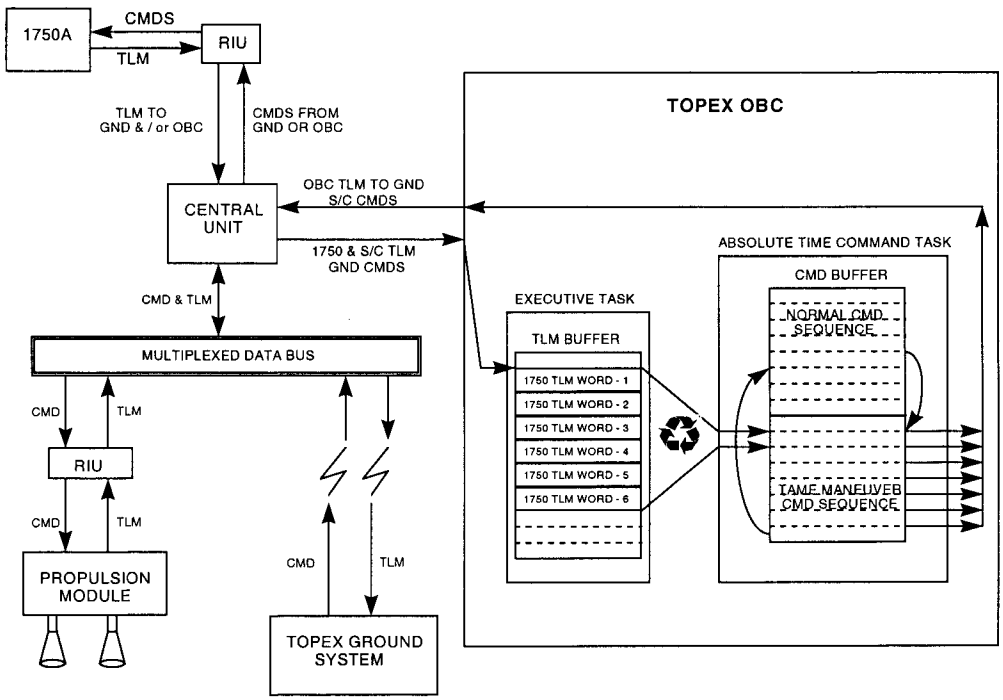

Figure 4: OBC-1750A data transfer architecture

Interacting Modules

Figure 5 shows the principal TAME modules and the data that are transmitted from one module to another. Functions of the principal modules are:

Planner—The planner searches for a violation-free attitude path that turns the spacecraft from its initial attitude to the burn attitude and, after the burn, returns the spacecraft to the attitude it would have at that time had no burn been performed. Violations that would cause an attitude path to be rejected are pointing the spacecraft's Z-axis too far from nadir, pointing one of two trackers too close to the sun, overheating, and exceeding the battery's discharge limit. The initial and final attitudes are on a yaw steering profile, which is based on the instantaneous geometry of the spacecraft, Earth, and Sun. As an aid in avoiding violations, the attitude path includes two intermediate attitudes, A and B (see figure 7). Attitude A is along the path from the initial attitude to the burn attitude, and B is along the return path following the burn. The turns account for turn-rate limits, settling times, and spacecraft dynamic behavior. The planner supplies the command translator and sequence generator the starting and ending attitude for each turn. Attitudes are supplied as quaternions. Thus, for the four turns, the planner supplies five quaternions. The attitude during the burn is constant, so no additional quaternions are required to define the entire attitude path.

Command translator—The command translator uses the quaternions supplied by the planner to compute a sequence of bias quaternions to be used by the sequence generator. The bias quaternions interpolate between the planner-supplied quaternions, thereby producing a sequence of uniformly spaced intermediate attitude points along any turn whose turn angle exceeds a lower limit.
Sequence Generator—The sequence generator merges the attitude path supplied by the planner and the command translator into a predefined sequence for preparing spacecraft hardware for the turning and thrusting maneuvers, and for restoring the coasting configuration. The predefined sequence involves pre-maneuver conditioning, such as starting catalyst-bed heaters, opening fuel latch valves, enabling propulsion module electronics, selecting thruster configuration, changing failure-detection limits, and others.

OBC—The OBC provides real-time telemetry data from other satellite systems such as the propulsion tank pressures to the 1750A. It also generates a time-tagged maneuver sequence based upon the sequence generator output and then executes the maneuver command sequence.

Physical Models—The physical models supply several types of data used for defining the planned attitude path. Pressures in tanks A and B, and ΔV (the velocity increment to be imparted), are inputs to the propulsion model for computing burn duration. In fact, the planner requests the tank pressures from the OBC, and writes them in its specification file; the propulsion module reads the pressures from the planner spec.

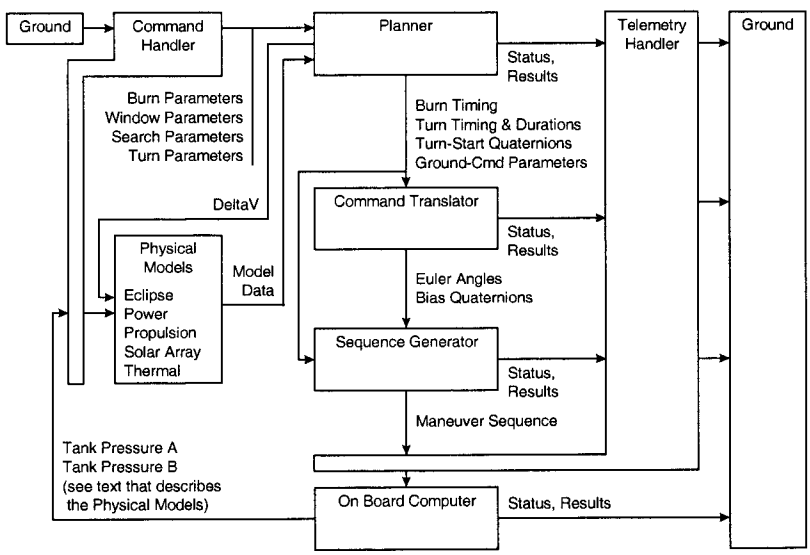

Figure 5: Flight Software Modules and Data Flow

Command Handler—The command handler receives commands from the ground and from the OBC, and routes the commands to the designated recipient. The command handler transmits to the ground via the telemetry handler an echo of every command received from the ground, and a copy of every command received from the OBC.

Telemetry Handler—The telemetry handler packs commands, status data, the maneuver sequence, and other results in a packing buffer, and transmits the buffer to the ground. It also transmits commands and the maneuver sequence to the OBC. Telemetry data comprises every type: string, boolean, integer, float, vector, matrix, and quaternion.

Intermodule Communication

Objects communicate with one another by means of flags and modes. Each flag or mode belongs to a single object, but each can be read by any object. The command handler commonly sets flags (but not modes) belonging to other objects; other objects rarely if ever set flags or other variables that do not belong to them.

Interacting Real-Time Processes

TAME comprises five real-time processes, as shown in Figure 6. The processes are (1) a main program, (2) a low-priority task, (3) a high-priority task, (4) a command handler, and (5) a telemetry handler.

The main program starts the two tasks, and then becomes inactive. The tasks cycle indefinitely, calling procedures in other objects but never being called. The command and telemetry handlers each provide one procedure that is driven by an interrupt stream (CommandInterface and TelemetryInterface), and numerous subprograms that are called from subprograms in the two tasks. Procedures CommandInterface and TelemetryInterface are independent of one another. The four processes that remain active are described in more detail in the following sections on the low- and high-priority tasks.

Low-Priority Task— The purpose of the low-priority task, is to execute compute-bound modules in the background, while the high-priority task services the StartInit command and downlink requests, which require immediate attention.

Following initialization, the task enters an endless loop, polling execution flags. The value of each execution flag, TRUE or FALSE, determines whether the task calls or does not call the corresponding module, i.e., the planner, command translator, or sequence generator.

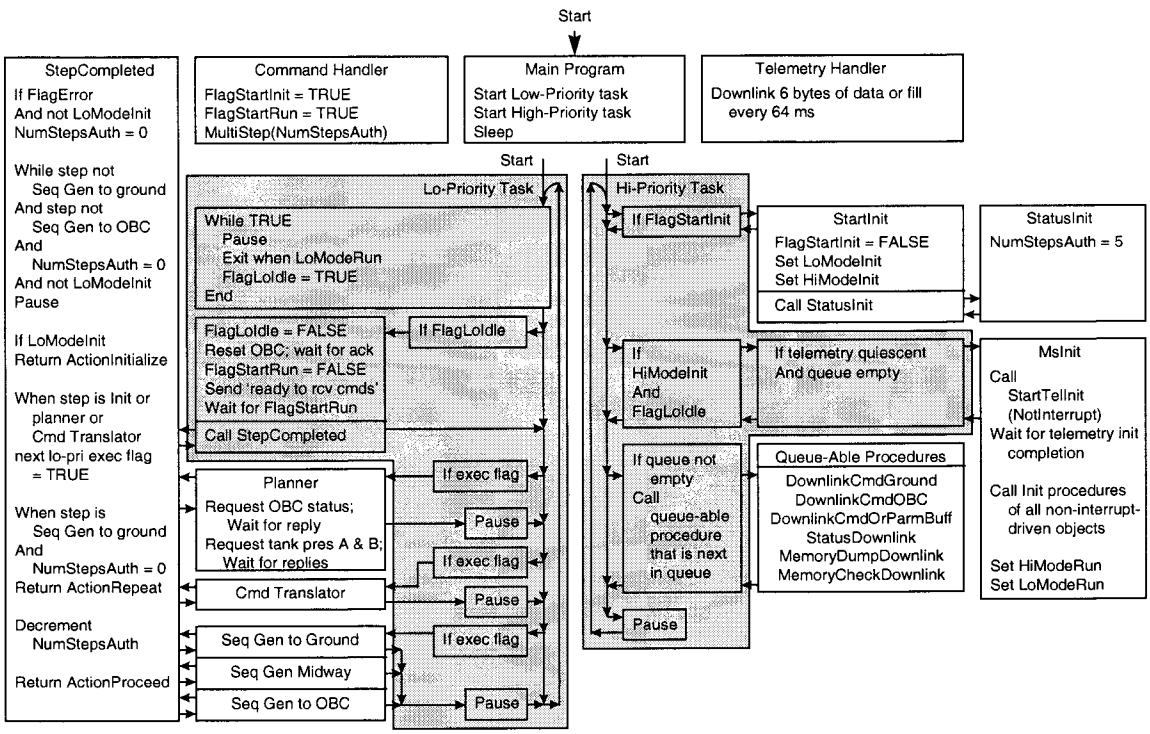

Figure 6: Flight Software's Real-Time Processes

Each execution flag is set only by procedure StepCompleted (See Figure 6). Each is initialized in its declaration to FALSE, reset to FALSE by the owning module's initialization procedure, and reset to FALSE by the procedure that is called to initiate the corresponding step.

The low-priority task's endless loop is divided into six *steps*. As the figure shows, the six steps are (1) Initialization, (2) planner execution, (3) command translator execution, (4) sequence generator telemetry to ground, (5) sequence generator midway, and (6) sequence generator telemetry to OBC. The code permits step allocations to be changed easily. The term "step" always refers to a bounded process in the low-priority task, never to one in the high-priority task.

Steps are controlled by MultiStep commands issued from the ground. Each of the steps ends with a call to the StepCompleted procedure, as the figure shows. The StepCompleted procedure, part of the STATUS object, receives (1) error status from each subprogram completing a step, (2) a further-steps-authorized number from the initialization process or from any MultiStep command, and (3) any StartInit command from the ground. On the basis of these data, StepCompleted sets or omits setting any execution flag for the succeeding step. On the final step, there is no execution flag for a succeeding step. Nonetheless, the StepCompleted procedure returns control to the caller, so that the caller can return control to the low-priority task, which then loops endlessly waiting for a StartInit command. In any case, StepCompleted returns an action directive to the caller.

The action directive that StepCompleted returns can be any one of three: initialize, repeat, or proceed. The initialize directive is returned in response to a StartInit command, which can be issued while any step is executing or while the low-priority loop is idling in the StartInit procedure. The repeat directive is returned only when the first of the three sequence-generator steps has executed. The continue directive is returned whenever a step completes without error and further step(s) are authorized.

The content of the action directive usually does not affect the response of the recipient. The low-priority task continues its polling loop regardless of directive content. The planner and command translator return control to the low-priority task regardless of directive content. Only the sequence generator responds according to directive content. Upon completing its first step, directive content causes the sequence generator to (1) repeat the step, (2) return control to the low-priority task in response to a StartInit command, or (3) proceed to its next (second) step. Upon completing its second step, the sequence generator's responses are (2) and (3) above. Upon completing its third step, the sequence generator returns control to the low-priority task regardless of directive content, the same as the planner and command translator.

High-Priority Task—The purpose of the high-priority task, is to service the StartInit command and downlink requests, which require immediate attention, while the low-priority task executes compute-bound modules in the background.

Following initialization at power-up, the task cycles endlessly polling for the StartInit flag and the presence of a queued downlink request. The StartInit flag can be set by the command handler in response to a StartInit command. Downlink requests are placed in the queue when the data for downlinking becomes available.

Planner Implementation

As mentioned earlier, the planner supplies five quaternions, shown in figure 7, defining spacecraft attitude at the start of the OMM sequence, at intermediate point A, during the burn, at intermediate point B, and at the finish of the OMM sequence. Each of the five quaternions defines spacecraft attitude with respect to the Orbit Reference Frame (ORF). The origin of the ORF is in the spacecraft, and the ORF rotates around the center of the Earth with the Z (yaw) axis always along the nadir, the X (roll) axis forward, the Y (pitch) axis along the negative of the orbital angular momentum. The quaternions for attitudes A and B are identical, making spacecraft attitude at A and B the same with respect to the ORF, but not the same inertially because of ORF rotation.

The search has four dimensions. These are the epoch of the burn centroid and the three Euler angles that define intermediate attitudes A and B. Burn epoch is the time of the midpoint of the burn with respect to the 6 AM epoch of the orbit, expressed in seconds. The 6 AM epoch is the time when the center of the Sun is in the spacecraft's local horizontal plane, and rising due to spacecraft orbital motion.

The search for a violation-free path is performed by procedure *planner*, one of 19 subprograms (procedures and functions) in the planner module. Additional subprograms are nested in a few of the 19. The search process, including interactions of procedure planner with the other subprograms, can be described in terms of procedure planner's code. The planner searches for a violation-free attitude path using five nested loops, as follows:

The two outermost loops search in the time dimension. The outer loop selects time windows within which burn epochs are permissible. The next nested loop tries burn epochs within the selected window. The three innermost loops set the Euler angles Phi, Eta, and Psi that define the intermediate attitudes A and B with respect to the ORF (see figure 8).

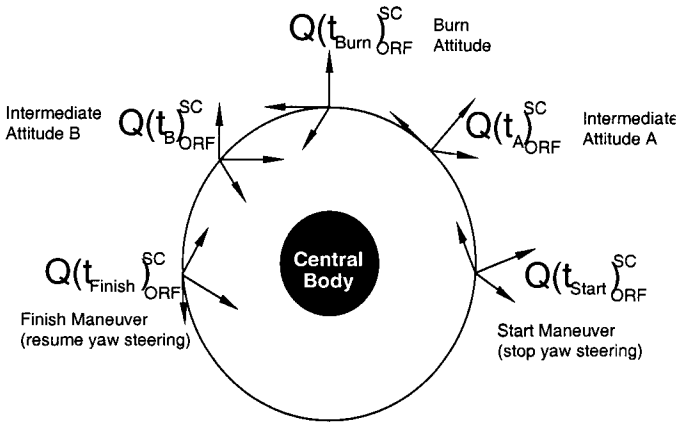

figure 7: Attitude Path and Intermediate Attitudes A and B

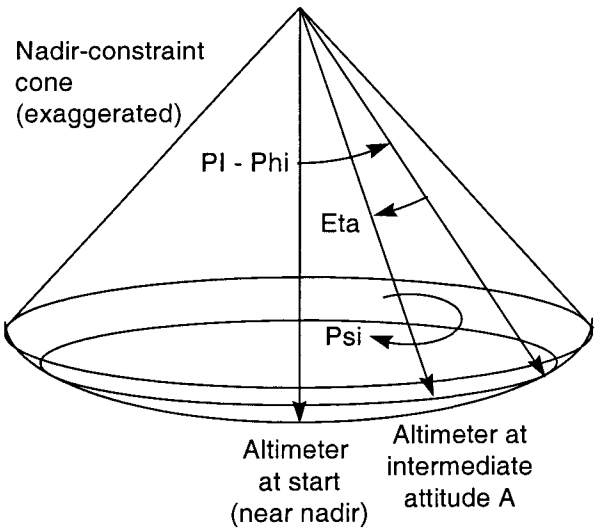

figure 8: Search Geometry and Euler Angles Phi, Eta Psi

All parameters used in the search are loaded by ground commands. Up to five windows and associated search parameters are permissible. Parameters include the step size for searching within a window, the three Euler angles, Burn parameters such as ΔV, turn parameters such as turn rates and settling times, and others.

Command Translator Implementation

As mentioned earlier, the bias quaternions computed by the command translator interpolate between the planner-supplied quaternions, thereby producing a sequence of uniformly spaced intermediate attitude points along any turn whose turn angle exceeds a lower limit. The method for computing the bias quaternions is chosen to match peculiarities of the Topex steering algorithms. Topex yaw steering is distinct from roll and pitch steering. The bias quaternions are intended to drive roll and pitch axes but not the yaw axis. The algorithm interpolates poorly when the pitch and roll angles are small compared to the yaw angle.

The bias quaternions are computed by interpolating only the roll and pitch Euler angles. The algorithm is:

1. For the roll and pitch axes, compute an intermediate Euler angle as initial + (final - initial) * i / n, where n is the number of steps, i is the step number in the range 1 .. n, and "initial" and "final" refer to the initial and final roll or pitch Euler angles. Thus, for each of the two axes, the first intermediate Euler angle is one step removed from the initial Euler angle, and the last intermediate Euler angle is identical to the final Euler angle.

2. For the yaw axis, set the intermediate Euler angle to zero.

3. Extract the bias quaternion from the product of the two direction cosine matrices that correspond to the pitch and roll intermediate Euler angles.

Sequence Generator Implementation

As mentioned earlier, the sequence generator (SEQGEN) produces an OMM sequence similar to the sequence generated using ground-based software tools. The architecture of the SEQGEN is described in

figure 9. The predefined sequence involves pre-maneuver conditioning, such as starting catalyst-bed heaters, opening fuel latch valves, enabling propulsion module electronics, selecting thruster configuration, changing failure-detection limits, and others.

SEQGEN has three primary software interfaces, the command translator, the database and the telemetry formatter. To simplify the design, SEQGEN uses 6 hard-coded templates, which describe the pre- and post-maneuver commands. These templates define the command and the relative time interval to the next command, but do not define the absolute time. This allows TAME software to produce a maneuver sequence for any desired time frame. The remaining inputs to the sequence are retrieved from the common data area. These inputs are uplinked from the ground or calculated by another module. SEQGEN must sort the inputs from the command translator based upon time, and integrate them into the predefined sequence. The integration process also involves checking and reorganizing the command sequence based upon spacecraft constraints. The outputs of SEQGEN consist of data that describe a single OBC command and timetag. These data are transferred to the telemetry Formatter for transmission to the OBC and to the ground.

When SEQGEN is called, it is given one of two destinations for the OMM sequence; the ground or the OBC. When the destination is ground, the OBC ignores the OMM sequence. In order to mitigate the risk of running the TAME experiment on an operational spacecraft, the calling procedure is written to enable calling SEQGEN in an infinite loop, with destination ground, until a MultiStep command is received authorizing further steps (see "MultiStep Commands" and "Risk Mitigation". Once additional steps are authorized, SEQGEN is called with destination OBC.

MultiStep Commands

The MultiStep concept, generalizing the single-step concept, is to uplink one or more commands authorizing TAME to take a designated number of further steps. The MultiStep command includes a single integer whose value designates the number of steps that TAME is authorized to take following completion of the step underway or already completed. If the value is zero, TAME idles until it receives a new MultiStep command or a StartInit. If a MultiStep command is received and its value is greater than zero, TAME takes the number of steps designated and then returns to idling. Thus the value can always be chosen to authorize TAME to complete all processing. The default for the number of steps authorized is chosen to enable TAME to run to completion. Its value is five, one less than the total number of steps that TAME can take, because it applies after the initialization step has been taken. The default is first set when TAME is first initialized, and it is restored each time TAME is reinitialized. Thus it is never necessary to uplink a MultiStep command if it is intended for TAME to run to completion.

5. DESIGN, DEVELOPMENT, INTEGRATION & TEST

The design and development process drove the integration and test of the TAME system. Therefore, before discussing the I&T process in detail, this section discusses the TAME design and development process, showing how it influenced the I&T process.

At high level, TAME development, shown in figure 10, consisted of the following steps:

Design and Develop TAME Algorithms—Algorithms for attitude path planning and sequence generation were originally developed partially in MATLAB and partially in FORTRAN. FORTRAN was used wherever functions were deemed suitable for reuse, e.g., propulsion and thermal models. New functions were developed in MATLAB.

Inherit and Rewrite 1750A Real-Time Modules—The architecture of the real-time processes (low- and high-priority tasks and interrupt-driven command and telemetry handlers) was substantially inherited. But all major modules, including the command and telemetry handlers, are completely new. All real-time modules were developed on a 1750A target computer in a stand-alone testbed.

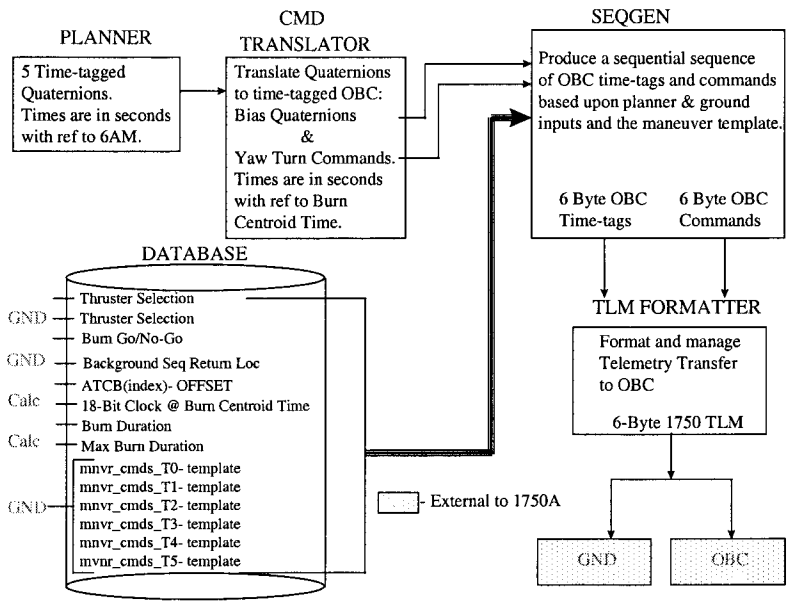

figure 9: Sequence Generator Data Flow

Translate and Convert TAME Algorithms—TAME algorithms were translated from their respective languages to Ada, producing the planner, command handler, and sequence generator. All modules were tested on a VAX workstation. Ada results were compared with MATLAB and FORTRUN results, and the algorithms were corrected in all three environments until all test results agreed to at least 13 significant places.

Integrate and Test 1750A Flight Software—Tame flight modules and real-time modules were integrated on the stand-alone test bed. Numerous test cases were rerun, and the results were compared with those from the VAX. Agreement was generally to about seven significant places.

Develop OBC Patch—The OBC patch algorithm was first developed in FORTRAN using an existing simulation system for unit test. The algorithm was then implemented in assembly language on the TOPEX system testbed. The patch was limited to only one section of memory and tests showed that running the patch had little chance of interfering with the normal operation of the TOPEX spacecraft, even if faults existed.

Integrate and Test 1750A and OBC Flight Software—All new flight software, 1750A and OBC, was integrated with existing flight software on the TOPEX system testbed. Tests proved that 1750A software and the OBC patch communicated correctly, and that executing the new software did not interfere with normal TOPEX operation.

Run Topex Autonomous Maneuver Experiments in Flight—The first of two flight experiments was performed November 17 1997, and deemed 100% successful. The remaining test consisted of performing a complete OMM and was performed successfully on December 1, 1998.

At each step in the development, realistic test cases were used to verify the planner, Table 1 shows the formal test cases. These test cases fall into two classes. The first class emulates previously conducted OMMs by restricting the planner's degrees of freedom. The spacecraft attitude is required to stay nadir pointed until just before the burn, and the only free variable is the burn epoch. In these test cases, the actual spacecraft telemetry is the "truth set" against which the TAME planner outputs are compared and evaluated. The other class of test cases are those which allow off-nadir pointing during the turn to burn attitude. These test cases take advantage of the planner's capability to "walk around" a constraint. The functionality and performance of TAME software in these cases is verified by analysis.

The final row of the table designates six project-approved acceptance-test cases. These cases are (1) using OMM 7 to verify TAME performance under nominal conditions; (2) using OMM 7 to verify that the satellite can enter safe hold mode when executing a TAME-generated sequence; (3) using OMM 7 to verify the MultiStep capability of preventing sending the OMM sequence to the OBC until authorized by a ground command; (4) verifying that the OBC can detect and avoid execution of an inaccurate or incomplete sequence; (5) verifying the planner's ability to detect that there is no violation-free solution, and to wait for further ground commands; and (6) using OMM 10 to verify TAME performance under nominal conditions.

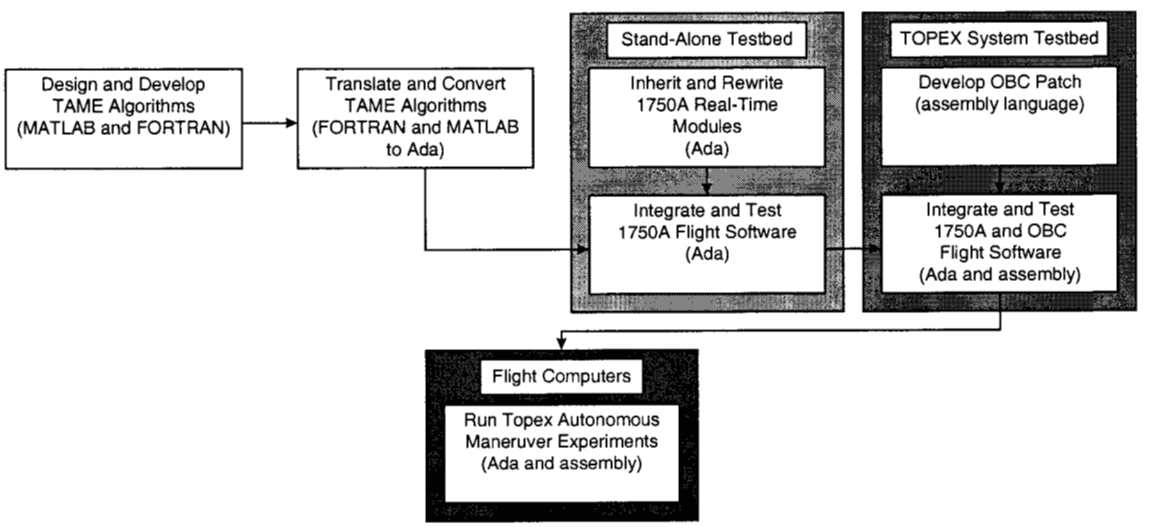

figure 10: TAME Development and Test Environments

	TEST ENVIRONMENT					
CASE(S)	MATLAB	VAX	STAND-ALONE TESTBED	TOPEX SYSTEM TESTBED	FLIGHT EXPERIMENT NUMBER	TRUTH SET
OMM 5, nadir	X	X	X	X		Flight data
OMM 6, nadir	X	X				Flight data
OMM 7, nadir	X	X	X	X		Flight data
OMM 8, nadir	X	X				Flight data
OMM 9, nadir	X	X				Flight data
OMM 10, nadir	X	X	X	X	1, 2	Topex testbed
OMM 10, 5^{O} off nadir	X	X	X	X	1	Topex testbed
Acceptance test cases (six)				X		Topex testbed

Table 1. Formal Acceptance Test Cases

6. Flight Experiments

To minimize risk to the satellite, and to allow sufficient confidence to be gained before executing a thrusting maneuver, two flight experiments were conducted.

Experiment Number One—involved loading of the 1750A software and running multiple solutions. This was executed successfully on November 17, 1997. Two OMM sequences were generated and telemetered to the ground. Both sequences were for the same ΔV request, but one produced a TOPEX type solution, maintaining nadir pointing throughout the maneuver, while the second demonstrated TAME's capability of turning off nadir to avoid constraint violations. Both sequences were tested in advance on the TOPEX system testbed. The OMM sequence was not transmitted to the OBC.

Experiment Number Two—exercised the complete end-to-end TAME process. This included generating a constraint free solution, planning the attitude maneuver, and implementing and executing the attitude and propulsive maneuvers. Further details regarding the flight experiment are provided later.

FLIGHT EXECUTION PREPARATIONS AND RESULTS

Extensive preparations were performed for the TAME flight execution. Preparations included executing a subset of the acceptance tests, performing training simulations, preparing command files, updating ground system databases and generating command and telemetry handling software tools. The TOPEX OMM execution date fell such that a propulsive maneuver was required for proper orbit maintenance. Hence, the TOPEX flight operations team prepared a ground based maneuver design in parallel with the TAME maneuver design. This parallel development effort allowed for another method of validating the TAME generated solution prior to use in flight. In addition to the testing, several peer reviews and flight readiness reviews were held to provide confidence to the flight team about TAME's capabilities and risk management strategies.

The current ground based maneuver design and execution process for TOPEX consists of a set of events occurring during a 3-4 week timeline. This timeline includes activities previously described required for design iteration and exchange of inputs and outputs between sub-system teams. Although the TAME process followed the same timeline, different activities were conducted during this timeframe. The initial TAME solution was produced in approximately 1 hour. One additional day was required to validate the solution using the TOPEX system testbed. The remainder of the time was spent tweaking the solution such that the outputs would be similar to the ground generated solution. This was necessary for the first in-flight execution to gain confidence in TAME and allow for interchangeability with the ground design as a contingency.

The following section presents some of the outputs from the TAME generated solution. Emphasis is given to showing TAME outputs that relate to the TOPEX mission constraints that the TAME solution was designed to meet. These variables are then compared with flight data from OMM10. In cases where flight telemetry data was not available, a comparison is provided based upon a prediction generated by the existing ground software.

The TAME generated roll, pitch and yaw angles are shown in figure 11 with the dashed lines representing nominal angles and the solid lines indicating attitude changes required or orienting the satellite prior to firing the thrusters. Prior to the propulsive maneuver, an attitude maneuver must be performed to properly orient the thrusters along the velocity vector. This is illustrated in figure 11 where TAME properly oriented the satellite while continuing to meet nadir pointing mission requirements. The majority of the rotation is performed in yaw with small biases in roll and pitch. The flight performance of the TAME design is illustrated in figure 12, which shows the flight yaw angle based upon telemetry during OMM10.

Another mission constraint is to not allow the sun to get closer than 20° Of the Star Tracker boresight.. The selected OMM10 date (12/1/98) presented a particularly difficult geometric problem for designing a maneuver solution while meeting the star tracker constraint. TAME quickly determined that no solution was feasible under these constraints. This task would normally take a couple of days of evaluation. Furthermore, TAME revealed that a 17° constraint may be feasible. With project approval, this constraint was modified to 17° for OMM10. The TAME solution (see Figure 13) shows the sun intrusion with a minimum sun angle of 17°. The flight data is shown for comparison in figure 14. Using the traditional design process took several days and much iteration to reach the same conclusion. This was a particularly valuable application of the TAME planner algorithm, which developed this solution within a span of 2-3 hours.

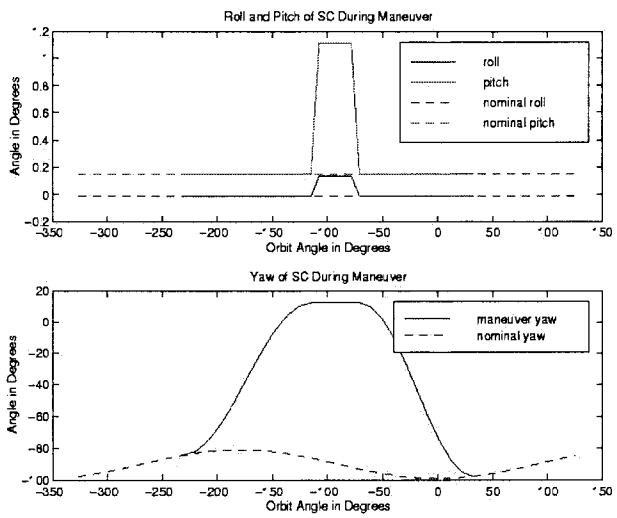

figure 11: TAME Generated Roll, Pitch, Yaw Angles

The primary thermal constraint during maneuvers relates to the TOPEX Microwave Radiometer (TMR). Predicting solar fluxes on the TMR provides the key decision component for meeting the thermal constraint. The TAME generated prediction is shown in figure 15. The flight data, which compare quite well, are shown in figure 16.

Battery State Of Charge (SOC), which provides an indication of the available energy, is a mission critical constraint. For TOPEX, the normal discharge levels during peak eclipse periods are approximately 12% (88% SOC). Hence, a mission guideline for maintaining similar charge levels is followed. The TAME generated solution (see Figure 17) predicted a SOC of 90.5% that is well within the mission guideline. The actual performance from the flight data (see figure 18) shows SOC to be consistent with the predict. The TAME power model does not account for shading of the solar panel or albedo, which accounts for the slight difference in discharge curves prior to reaching the minimum SOC. To ensure sufficient memory margin, TAME's physical models were simplified to include the absolute minimum required to accomplish the tasks. Operationally, TOPEX limited OMM maneuvers to orbits with β' between 30° to 70°. A high β' maneuver, such as the OMM 10, was never needed. With that in mind, high β' conditions, such as shading effects, were not incorporated in the TAME power module design. This is another testimony to the TAME robust design that it can perform so well under such diverse conditions.

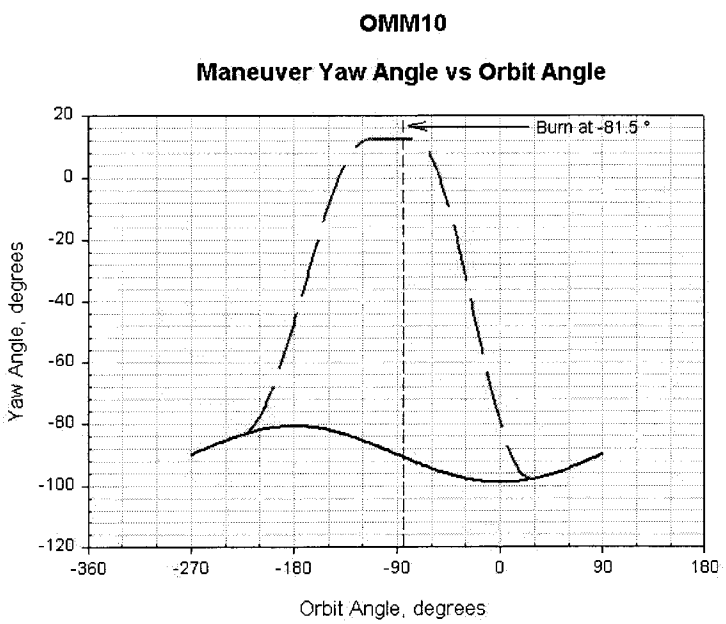

figure 12: OMM10 Flight Roll, Pitch, Yaw Angles

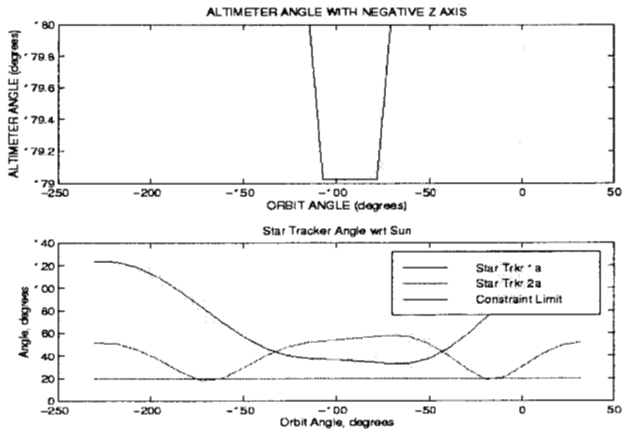

Figure 13: TAME predicts Altimeter and Star-Tracker Angles for OMM10.

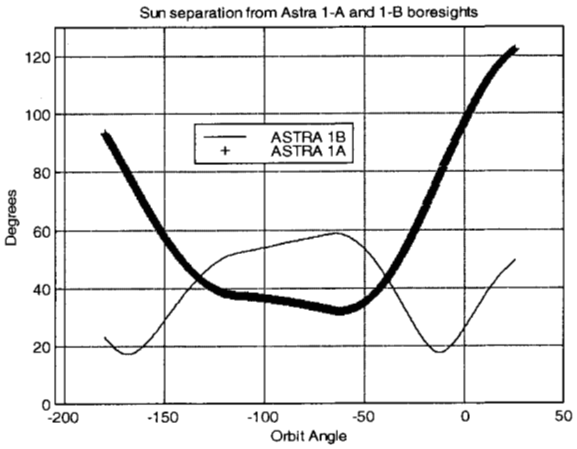

figure 14: Flight data of Star Tracker and sun separation angles

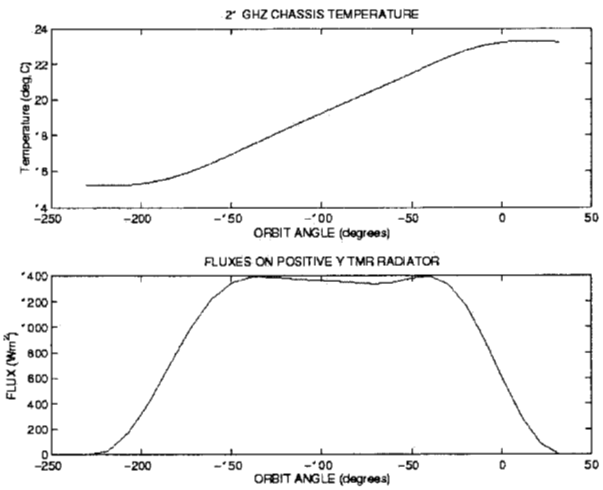

figure 15: TAME generated Thermal Variables for OMM10

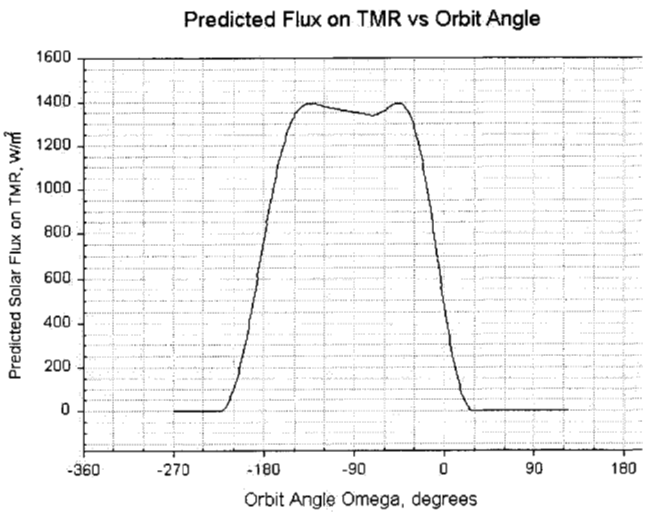

figure 16: Predicted Solar Flux on TMR for OMM10

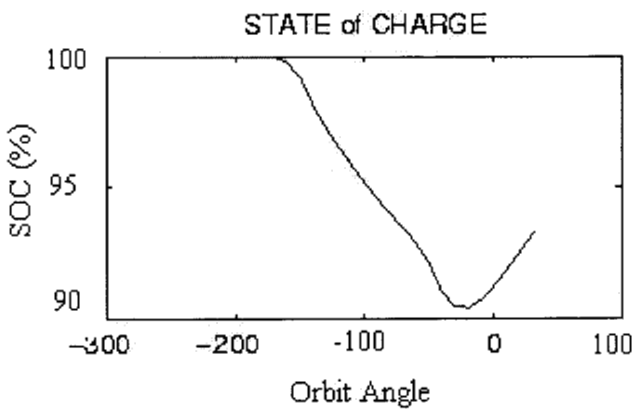

Figure 17: TAME Predict for Battery State Of Charge

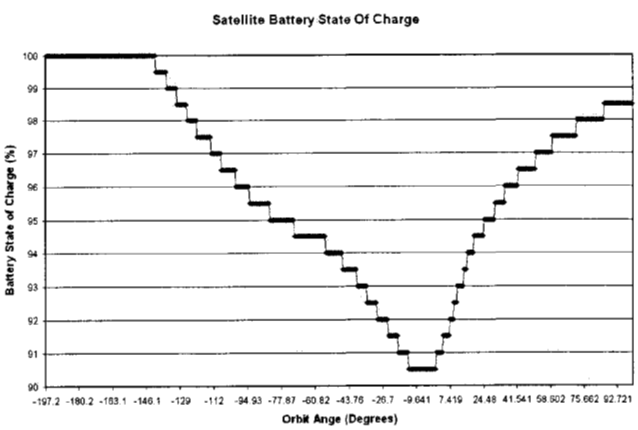

figure 18: OMM10 Flight Battery State Of Charge

Risk Mitigation

For TAME to be implemented and executed on a existing flight program significant emphasis was placed on risk mitigation. Some of these provisions are optional, controlled by ground commands intended to be exercised only as an additional verification for the first complete flight experiment. The specific steps to mitigate risks are described below.:

- The MultiStep system in the 1750A enables ground operators to verify TAME results step by step.
- The OMM sequence is not transmitted to the OBC until it has been transmitted to the ground, it has been verified, and a ground command has been received to authorize transmitting the sequence to the OBC.
- Numerous checks, such as acknowledgment and handshaking, are made in transmitting data from 1750A to OBC.
- The OBC rejects invalid or incomplete data, returns one of several error flags to the 1750A and the ground and terminates execution of the TAME OBC code.
- Upon receipt of an error flag from the OBC, TAME issues an error flag to the ground, and ceases execution until it receives further commands from the ground.
- Even when transmission of the OMM sequence has passed all OBC and 1750A checks, the OBC does not execute the sequence until authorized to do so by a ground command.
- Ground commands can load a jump command in the OBC, causing the OBC to skip the burn while executing the remainder of the OMM sequence.
- The OBC code modifications for TAME are limited to one specific area and one processor task. Hence, an error would impact a non-critical process and prevent the satellite from entering a safe-ing condition.
- Many additional GO/NOGO switches are implemented in OBC code.

7. LESSONS LEARNED

Throughout the development and implementation of TAME numerous valuable lessons were learned applicable to various areas not limited to just autonomy. Some of the lessons are captured below:

- Autonomy capabilities should be considered during early stage of flight program development.
- It is critical to ensure generic model design with maximum flexibility to alter design parameters
- For portability, all software modules should have structured interfaces with generic inputs and outputs.
- Dedicated hardware resources for autonomy may not be required if sufficient margins for computing resources exist.
- Maximum utilization of ground simulation and modeling should be utilized to avoid in-flight maintenance.
- Autonomy tools can be utilized for easy ground analysis programs.

8. CONCLUSIONS

The TOPEX Autonomous Maneuver Experiment (TAME) has demonstrated the feasibility of using on-board algorithms for performing autonomous attitude and propulsive maneuvers. TAME has successfully displayed the capabilities of the maneuver planner algorithm under real mission constraints and oversights required to ensure the safety of the spacecraft. TAME has also demonstrated the time and resource efficiencies that can be obtained from using on-board autonomy.

The TAME algorithms could be utilized in any aspect of spacecraft operation that requires the planning and implementation of constraint free turns for spacecraft or platform orientation. Using on-board orbit determination along with the TAME algorithm constitute a complete end-to-end autonomous maneuver process.

ACKNOWLEDGMENTS

The research described in this paper was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

REFERENCES

[1] Aljabri, A. S., Kia, T., and Lai, J. Y., "Highly-Autonomous Event-Driven Spacecraft Control," *IAA International Conference on Low-Cost Planetary Missions*, Maryland, April 1994.

[2] T. Kia, A. Aljabri, R. Goddard, T. Munson, G. Kissel, H. Lin, P. Vaze, "TOPEX/POSEIDON Autonomous Maneuver Experiment (TAME)," 19th Annual AAS Guidance and Control Conference, February 7-11, 1996 Breckenridge, CO.

[3] T. Kia, J. Mellstrom, A. Klumpp, T. Munson & P. Vaze, "TOPEX/POSEIDON Autonomous Maneuver Ex-

periment (TAME) Design and Implementation," 20th Annual AAS Guidance and Control Conference, February 5-9, 1997 Breckenridge, CO.

[4] Dennehy, C. J., Ha, K., Welch, R. and Kia, T. "On-board Attitude Determination for the TOPEX Satellite," AIAA Guidance, Navigation and Control Conference, Boston, MA, August 1989.

[5] Bhat, R. S., Frauenholz, R. B. and Canneli, P. E. "TOPEX/POSEIDON Orbit Maintenance Maneuver Design," AAS/AIAA Astrodynamics Specialist Conference, Stowe, Vermont, August 1989.

BIOGRAPHY

Parag Vaze is a project element manager in the flight systems section at the Jet Propulsion Laboratory (JPL). He has worked in mission operations for the Topex/Poseidon project for 8 years. He has developed and tested hardware and software for ground and flight space systems at JPL, Orbital Sciences Corp and GTE-Spacenet. He holds a BSEE from University of Pittsburgh and is currently pursuing a master's in computer science at Azusa Pacific Univeristy.